Pergamon

Computers Math. Applic. Vol. 35, No. 7, pp. 45–54, 1998

0898-1221/98 $19.00 + 0.00

PII: S0898-1221(98)00031-5

Efficient Parallel Multigrid Based Solvers for Large Scale Groundwater Flow Simulations

F. SAIED
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.
saied@cs.uiuc.edu

G. MAHINTHAKUMAR
Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6203, U.S.A.
kumar@ornl.gov

Abstract—In this paper, we present parallel solvers for large linear systems arising from the finite-element discretization of three-dimensional groundwater flow problems. We have tested our parallel implementations on the Intel Paragon XP/S 150 supercomputer using up to 1024 parallel processors. Our solvers are based on multigrid and Krylov subspace methods. Our goal is to combine powerful algorithms and current generation high performance computers to enhance the capabilities of computer models for groundwater modeling. We show that multigrid can be a scalable algorithm on distributed memory machines. We demonstrate the effectiveness of parallel multigrid based solvers by solving problems requiring more than 64 million nodes in less than a minute. Our results show that multigrid as a stand alone solver works best for problems with smooth coefficients, but for rough coefficients it is best used as a preconditioner for a Krylov subspace method.

Keywords—Hydrology, Multiprocessors, Numerical methods, Partial differential equations, Multigrid method.

1. BACKGROUND

In order to determine flow fields in a groundwater aquifer, a partial differential equation (p.d.e.) commonly referred to as the groundwater flow equation needs to be solved. For the steady-state saturated case, this equation is an elliptic p.d.e. given by

$$\nabla \cdot (K \nabla h) - q = 0, \tag{1}$$

where K is the hydraulic conductivity tensor, h is the head field, and q represents the source/sink terms coming from injection/pumping wells. In general, finite-element or finite-difference techniques are used to discretize equation (1).

This work was sponsored by the Center for Computational Sciences of the Oak Ridge National Laboratory managed by Lockheed Martin Energy Research Corporation for the U.S. Department of Energy under contract number DE-AC05-960R22464. The authors also acknowledge the use of the Intel Paragon XP/S 150 computer, located in the Oak Ridge National Laboratory Center for Computational Sciences (CCS), funded by the Department of Energy's Mathematical, Information, and Computational Sciences (MICS) Division of the Office of Computational and Technology Research.

For many realistic problems, the groundwater flow equation involves rough coefficients (tensor K) resulting from heterogeneous hydraulic conductivity fields (or K-fields). In order to resolve fine-scale heterogeneity effects on large-scale regional models (e.g., in the order of kilometers) a fine discretization is required (e.g., in the order of few meters). For such problems, finite-element or finite-difference discretizations give rise to very large linear systems (in the order of tens of millions) that need to be solved.

The matrices that result from the discrete approximation of equation (1) are sparse, symmetric, and positive definite. The preconditioned conjugate gradients are a popular Krylov method (see next section) commonly used to solve such systems [1,2]. For methods such as preconditioned conjugate gradients, the number of iterations required for convergence increases with the problem size and the degree of heterogeneity when traditional preconditioners such as diagonal scaling or incomplete Cholesky are used. However, we can improve on this behavior by using a multigrid method, either on its own, or as a preconditioner in a Krylov subspace method. By using multigrid techniques, we can make the convergence behavior less dependent on the problem size and the roughness of the coefficients [3–6]. But the difficulty in implementing multigrid techniques on distributed memory machines has prevented this method from gaining popularity on machines such as the Intel Paragon.

In this work, we implement parallel multigrid based solvers on the Intel Paragon and compare their performance with diagonally preconditioned conjugate gradients. Performance is measured in terms of raw solution time, scalability, parallel efficiency, and Megaflop rate (a Megaflop/s stands for 10^6 floating point operations per second). Efficiency of multigrid methods for increasing problem sizes and increasing roughness is also compared.

2. KRYLOV SUBSPACE METHODS

Krylov subspace methods for solving a linear system $Ax = b$ are iterative methods that pick the j^{th} iterate from the following affine subspace:

$$x_j \in x_0 + K_j(A, r_0),$$

where x_0 is the initial guess, r_0 the corresponding residual vector, and the Krylov subspace $K_j(A, r_0)$ is defined as

$$K_j(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{j-1}r_0\}.$$

These methods are very popular for solving large sparse linear systems because they are powerful and yet offer considerable savings in both computation and storage. Some of the more popular Krylov methods are Preconditioned Conjugate Gradients (PCG), Bi-Conjugate Gradient Stabilized (Bi-CGSTAB), Generalized Minimal Residual (GMRES), Quasi-Minimal Residual (QMR), and Adaptive Chebychev [7–9]. Of these, PCG is used for only symmetric positive definite systems.

3. MULTIGRID METHODS

Multigrid methods for partial differential equations use multiple grids for resolving various features of the solution on the appropriate spatial scales [10–13]. They derive their efficiency by not attempting to resolve coarse scale features on the finest grid.

The basic idea of multigrid is depicted in Figure 1, for the two-grid version. Starting with an initial guess, u_h^{old}, on the finest grid, we apply ν_1 iterations of a smoothing method (R_h), such as weighted Jacobi or Gauss-Seidel and form the residual r_h of the resulting grid vector. This is "restricted" down to the coarse grid, where it is used as the right-hand side (r_{2h}) of the coarse grid correction equation, $L_{2h}c = r_{2h}$, where L_{2h} is an appropriately defined coarse grid operator. The solution to this problem (c_{2h}) is interpolated back to the fine grid where it is added to the current approximation. Finally, an additional ν_2 sweeps of the smoother are

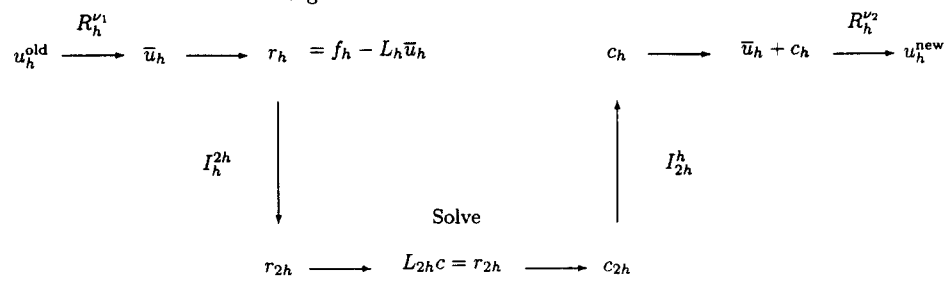

Figure 1. The two-grid version of multigrid.

applied to the corrected approximation, to obtain u_h^{new}. The grid transfers involve fine to coarse (restriction, I_h^{2h}) and coarse to fine (prolongation or interpolation, I_{2h}^h) stages. At the coarsest level, a full matrix solve is performed before moving up to the next finer level. The coarse-grid solve is usually done by PCG or banded Gaussian elimination.

In practice, the two-grid algorithm is applied recursively. The most common approach is the V-cycle, where an initial guess must be supplied on the finest grid. The V-cycle can be used on its own or as a preconditioner to a Krylov method. The performance of multigrid can be "tuned" through an appropriate choice of parameters like the number of levels, or the smoothing sweeps (ν_1, ν_2).

4. ALGORITHMIC FRAMEWORK

For the three-dimensional isotropic case, equation (1) reduces to

$$\frac{\partial}{\partial x}\left(K(x,y,z)\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K(x,y,z)\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K(x,y,z)\frac{\partial h}{\partial z}\right) = q, \tag{2}$$

where $K(x,y,z)$ is the hydraulic conductivity value at location (x,y,z). To solve equation (2), we employ the Galerkin finite element discretization using eight-node linear brick elements [14,15]. This discretization results in a matrix equation of the form $Ax = b$, where A is a sparse, symmetric positive definite matrix. For a rectangular grid structure and "natural ordering" of unknowns, matrix A has a 27-diagonal banded nonzero structure. In our implementation, we exploit symmetry and store only the 14 super-diagonals of the matrix.

For the multigrid implementation, we use a V-cycle for each multigrid iteration. In order to construct the restriction operator within each V-cycle, we implemented three methods: simple injection, half weighting (7-point), and full weighting (27-point). For the prolongation (interpolation) operator within each V-cycle, we used a linear interpolation scheme. The coarse grid operator for each level is simply the finite-element global matrix at these levels. For cases with rough coefficients, the elemental hydraulic conductivity values at the coarser levels are obtained by a local averaging scheme. We implemented three options to perform this averaging: arithmetic, geometric, and harmonic averaging. For most of our test cases, simple injection and arithmetic averaging proved to be the best options. For the coarse grid solve we used the diagonally preconditioned conjugate gradient method.

For the smoothing operation, we chose the weighted (or underrelaxed) Jacobi, which, for $Ax = b$ and $A = D - L - U$, is defined by

$$x^{(n+1)} = \left[(1-\omega)I + \omega D^{-1}(L+U)x^{(n)} + \omega D^{-1}b,\right.$$

where ω is the weighting factor. Although Jacobi is less powerful than methods such as Gauss-Seidel, it is easily parallelized and is generally adequate as a smoother.

We also implemented options to use multigrid as a preconditioner for CG and BiCGSTAB methods. Summarizing, our parallel solvers consisted of the following methods: DPCG (diagonally

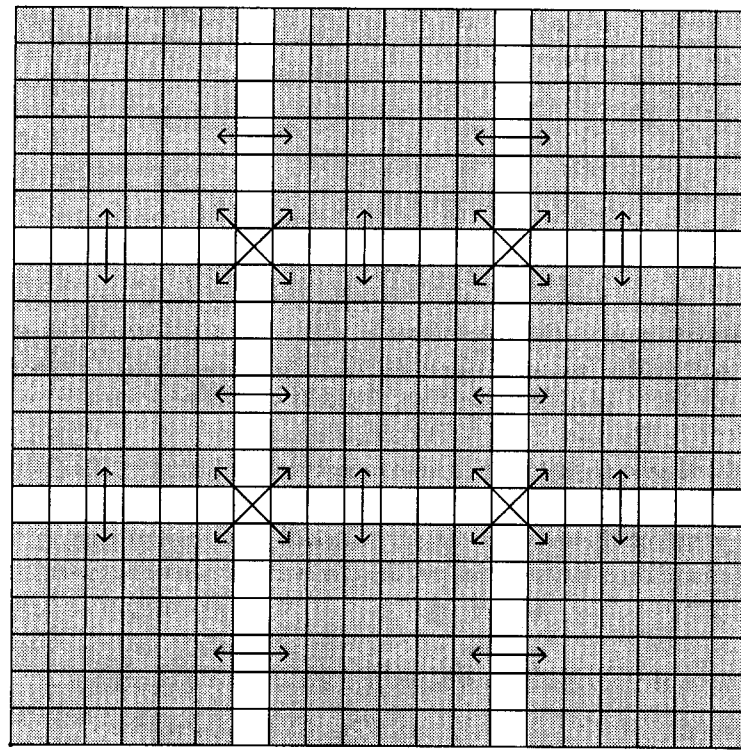

Figure 2. Plan view of two-dimensional domain decomposition. Each gray region belongs to a processor; the white regions are overlapped. The arrows show the communication pattern.

preconditioned conjugate gradients), MG (stand alone multigrid solver), MGCG (multigrid preconditioned conjugate gradients), and MGBiCGSTAB (multigrid preconditioned Bi-CGSTAB). The results for BiCGSTAB with multigrid preconditioning were very similar to those for MGCG, and will not be presented here.

5. PARALLEL IMPLEMENTATION

We used the Intel Paragon XP/S 150 (1024 MP-nodes) at the Oak Ridge National Laboratory's Center for Computational Sciences (CCS)[1] for our parallel implementation. XP/S 150 has 1024 MP (multiple thread) nodes connected by a 16 × 64 rectangular mesh configuration. Each node has a local memory of 64 Megabytes. The native message passing library on the Paragon is called nx. The inter-node message bandwidth is about 152 Mb/s for long messages ($\approx$ 1 Mb) with a zero-length latency of 35 ms.

For parallelization, we used a two-dimensional (2-D) domain decomposition in the x and y directions as depicted in Figure 2. A 2-D decomposition is generally adequate for groundwater problems because common groundwater aquifer geometries involve a vertical dimension which is much shorter than the other two dimensions. For the finite-element discretization such decomposition involves communication with at most 8 neighboring processors. We note here that a 3-D decomposition in this case would require communication with up to 26 neighboring processors.

We overlap one layer of processor boundary elements in our decomposition to avoid additional communication during the assembly stage at the expense of some duplication in element computations. There is no overlap in node points. In order to preserve the 27-diagonal band structure

[1]For more details, see the CCS web page at `http://www.ccs.ornl.gov`.

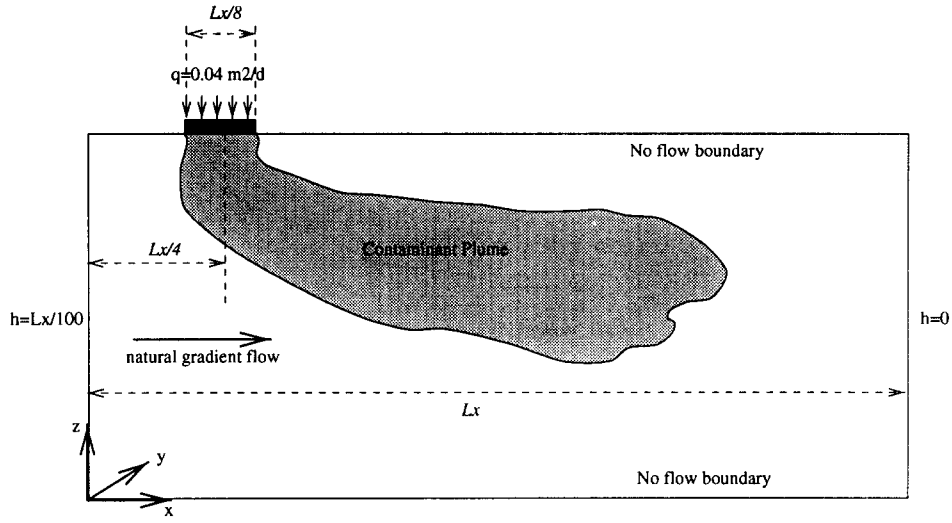

Figure 3. Vertical cross-section of model problem.

within each processor submatrix, we perform a local numbering of the nodes for each processor subdomain. This resulted in noncontiguous rows being allocated to each processor in the global sense. For local computations each processor is responsible only for its portion of the rows which are locally contiguous. However, such a numbering gives rise to some difficulties during explicit communication and I/O stages. For example, in explicit message passing, noncontiguous array segments had to be gathered into temporary buffers prior to sending. These are then unpacked by the receiving processor. This buffering contributes somewhat to the communication overhead. When the solution output is written to a file, we had to make sure that the proper order is preserved in the global sense. This required noncontiguous writes to a file resulting in I/O performance degradation, particularly when a large number of processors were involved.

For simplicity, we use the same static decomposition at all multigrid levels. This strategy limits the number of multigrid levels that can be used because even the coarsest grid problem has to be distributed across all processors.

All explicit communications between neighboring processors were performed using the Paragon's asynchronous nx calls. System calls were used for global communication operations such as those used in dot products. An MPI (Message Passing Interface) version of the code has been implemented and tested on a variety of parallel architectures [16]. The codes are written in FORTRAN using double-precision arithmetic. Although each MP node on the XPS/150 is capable of using up to three parallel threads, the results presented in this paper are only for the single threaded mode.

6. MODEL PROBLEM

For all the test simulations, we setup a model problem as shown in Figure 3. This setup corresponds to a contamination scenario where the contaminant leaches from a single rectangular source into a naturally flowing groundwater aquifer.

The flow field generated from such simulation can be used as an input to a transport simulator to generate the contaminant plume [1]. Boundary conditions for this setup are as follows: Fixed heads of $h = Lx/100$ and $h = 0$ at the faces of $x = 0$ and $x = Lx$, respectively, a rectangular patch of $Lx/8 \times Ly/8$ centered at $(x = Lx/4,\ y = Ly/2,\ z = Lz)$ with a uniformly distributed flux of 0.04 m2/d, and no flow boundaries elsewhere. For tests involving heterogeneous K-fields (i.e., rough coefficients), we obtained the spatially correlated random K-fields by using a parallelized version of the turning bands code [16]. The degree of heterogeneity is measured by the parameter σ, which is an input parameter to the turning bands code.

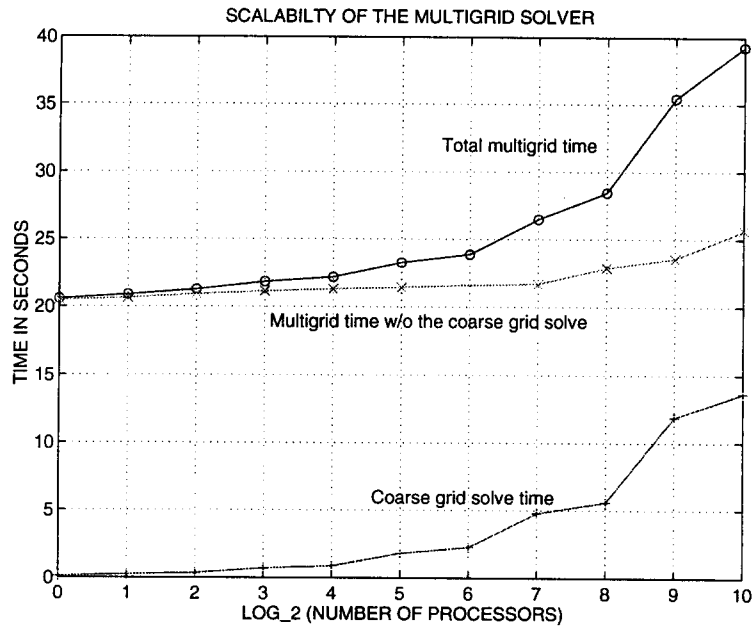

Figure 4. Scaling behavior of multigrid.

7. PERFORMANCE RESULTS AND DISCUSSION

In this section, we present and compare the performance of our implementations with respect to problem size, scalability, raw floating point performance, and roughness of coefficients. The following selections were used for all performance tests unless otherwise stated:

- convergence criteria for matrix solution: two-norm of relative residual $< 10^{-8}$,
- coarse grid solve: DPCG with tolerance set to 10^{-4},
- homogeneous K-field (constant coefficient case),
- timings are for matrix solution only.

In the following, P denotes the number of processors. Timings were obtained by the Paragon's dclock() system call. Timings reported are for the processor that takes the maximum time.

7.1. Scalability of Multigrid and DPCG

We analyze the scalability of multigrid and DPCG by increasing the problem size with a corresponding increase in the number of processors (i.e., N/P is fixed). The results of this experiment are presented in Table 1. The grid sizes ranged from $33 \times 33 \times 65$ for a single processor to $1025 \times 1025 \times 65$ for the 1024 processors. The most striking result in Table 1 is that the multigrid iterations remain fixed, while the DPCG iterations grow as we scale up the problem size. Furthermore, we see that the multigrid solution time for the largest problem (approximately $68\,M$ nodes) on 1024 processors is about twice that for the smallest problem (approximately $70\,K$ nodes) on 1 processor. In particular, the 68 million node problem was solved in under 40 seconds on 1024 processors.

The multigrid data from Table 1 is plotted in Figure 4. The total time is broken down into the coarse grid solve time, and the rest. A closer inspection of our timings revealed that most of the loss in scalability is due to the coarse grid solve which is performed by DPCG. Even though the multigrid iterations remain the same throughout the scaling process, the DPCG coarse grid solve iterations increase because the coarse grid problem becomes larger as we scale. By the same token, we can see from Figure 4 that all phases of the V-cycle other than the coarse grid solve show very good scalability.

Table 1. Scaling behavior of multigrid and DPCG. Homogeneous K-field, four grid levels, and $\nu_1 = \nu_2 = 3$. P is the number of processors.

P	$Px \times Py$ ($= P$)	$nx \times ny \times nz$	MG Iter	MG Time	DPCG Iter	DPCG Time
1	1 x 1	33 x 33 x 65	6	20.56	98	42.3
2	2 x 1	65 x 33 x 65	6	20.87	147	61.0
4	2 x 2	65 x 65 x 65	6	21.26	151	63.6
8	4 x 2	129 x 65 x 65	6	21.83	245	98.7
16	4 x 4	129 x 129 x 65	6	22.18	242	98.4
32	8 x 4	257 x 129 x 65	6	23.26	358	142
64	8 x 8	257 x 257 x 65	6	23.87	429	171
128	16 x 8	513 x 257 x 65	6	26.48	664	260
256	16 x 16	513 x 513 x 65	6	28.47	756	299
512	32 x 16	1025 x 513 x 65	6	35.45	1203	469
1024	32 x 32	1025 x 1025 x 65	6	39.28	1612	670

7.2. Parallel Performance for Fixed Problem Size

In Figure 5, we compare the parallel efficiency of the total time to the matrix solution and explicit interprocessor communication times. Timings are for the fixed size problem ($257 \times 257 \times 65$) using the MG solver. The number of levels was three and $\nu_1 = \nu_2 = 3$. The total time includes initial setup, finite-element matrix assembly, matrix solution, and I/O. From Figure 5, we can observe that even though the MG solution has subpar parallel efficiency, the total time has a reasonable speed up behavior. The explicit communication time decreases slightly in the beginning and then starts to gradually increase as we increase P. We attribute the initial drop in communication time to messages becoming shorter (message bandwidth limited) and the increase near the end to the latency overhead.

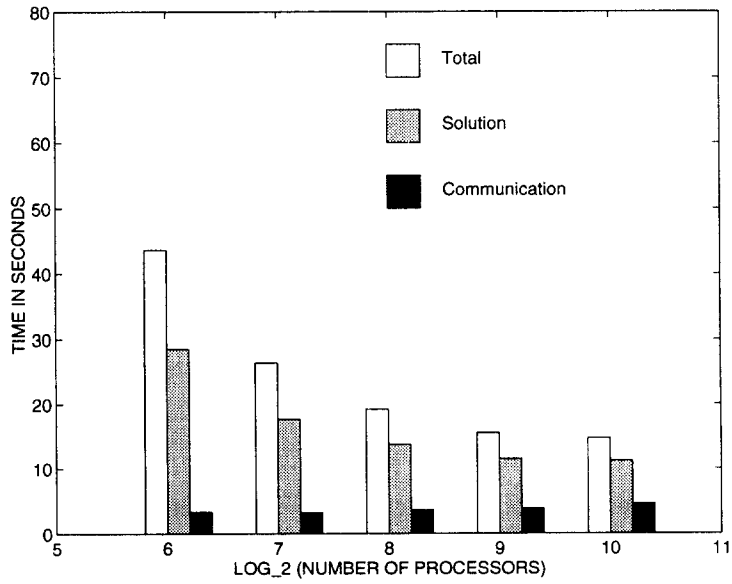

Figure 5. Parallel performance for fixed problem sixe ($257 \times 257 \times 65$). The number of multigrid levels was three and $\nu_1 = \nu_2 = 3$.

7.3. Roughness of Coefficients

The roughness of the coefficients of equation (2) is measured by a parameter σ which represents the degree of heterogeneity of the K-field. In Table 2, we show the effect of increase in σ on the

convergence behavior of our solvers. The results we present are for a $1025 \times 1025 \times 65$ problem on 1024 processors. The multigrid-based methods used five levels and five pre- and postsmoothings. The results show that multigrid is best used as a preconditioner when the heterogeneity is high. Examining Table 2 reveals that the convergence of MGCG is less affected by σ than MG. This is interesting because we did not use operator-based restriction and prolongation for the multigrid methods in our implementation. We believe this is related to the robustness of our coarse grid operator which is the based on coefficient averaging and a finite element discretization.

Table 2. Effect of varying heterogeneity. 1024 processors, $1K \times 1K \times 64$ mesh. The numbers in the table are times in seconds, and in parentheses, the number of iterations.

	MG	DPCG	MGCG
homog.	34.81 (4)	875.93 (2035)	44.78 (4)
heterog. ($\sigma = 1.0$)	65.47 (8)	957.55 (2225)	61.44 (6)
heterog. ($\sigma = 2.0$)	129.74 (16)	1132.76 (2633)	85.99 (9)

7.4. Floating Point Performance

We estimated the Mflop rates for our solvers using a MATLAB routine which computes the number of floating point operations as a function of various V-cycle parameters. The peak performance for the MG solver is about 4.2 Gflops compared to 10.3 for DPCG. These numbers are for the largest problem shown in Table 1. For the MG solver, the Mflop per processor ranged from 7.8 for the single processor problem in Table 1 to 4.1 for the largest problem on 1024 processors. For double precision floating point operations involving sparse matrices, 15 Mflops per processor is usually considered very good for the Portland Group Fortran compiler on the i860 chip.

7.5. Tuning the Performance of Multigrid

The performance of multigrid solvers can be tuned by varying parameters that control the multigrid V-cycle. For example, by selecting optimal values for the number of smoothings and the number of levels, we can improve the performance of the solver for a given problem size and processor count.

In Figure 6, the effect of varying (ν_1, ν_2) is examined, for the homogeneous case. Recall that ν_1 and ν_2 are the number of presmoothings and postsmoothings, respectively. For this experiment, we chose $\nu_1 = \nu_2$. N/P, the number of unknowns per processor, was kept fixed for all the cases, $P = 1$, $P = 256$, and $P = 1024$. Note that the pay-off for doing more smoothings is greater for $P = 1024$ than for the single processor case. The reason is that the number of V-cycles, and hence, the number of coarse grid solves is reduced, as ν_1, ν_2 are increased. This reduces the impact of the coarse grid solve which is the least efficient component of the parallel multigrid algorithm.

We also studied the performance of the code by varying the number of levels used in the multigrid algorithm. We note here that our code is limited to five levels on the Paragon, because we require the coarse grid problem to be distributed across all processors. Although we do not present the results here, for large problems, it pays to use all five levels because this cuts down the fraction of the time spent in the coarse grid solver. However, the improvement in time decreased as we increased the number of levels (e.g., the improvement in time by going from four to five levels is less than that going from three to four levels). This implies that by going beyond five levels at the expense of additional coding and load imbalance, overhead may not appreciably improve the performance.

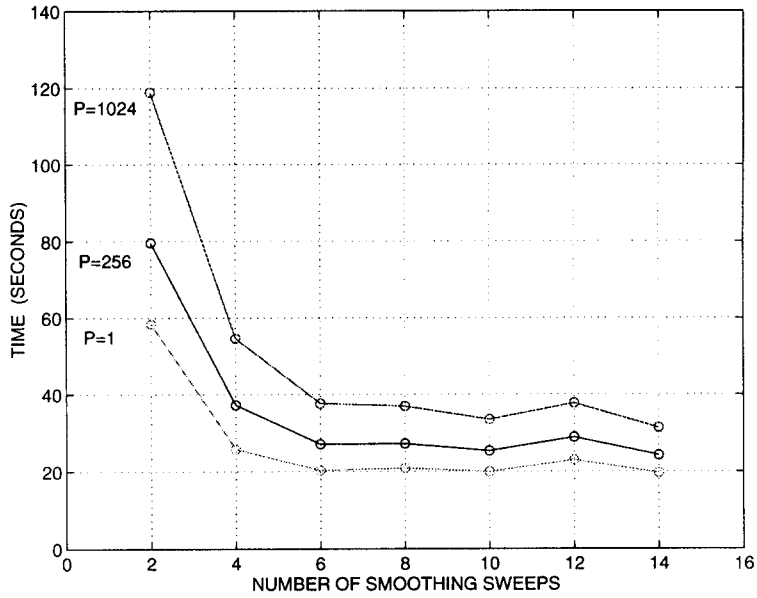

Figure 6. Effect of varying (ν_1, ν_2). For this experiment we chose $\nu_1 = \nu_2$. N/P was kept fixed for all three cases, $P = 1$, $P = 256$, and $P = 1024$.

8. CONCLUSIONS

We have implemented multigrid for the solution of the finite-element equations for the 3-D groundwater flow problem on distributed memory machines. Of the solvers we have implemented, we conclude that multigrid solvers are the most efficient for solving very large groundwater flow problems on the Paragon. For example, for the $1K \times 1K \times 65$ node problem, DPCG would have to run at 150 Gflops to solve the problem as quickly as multigrid. The performance of our multigrid solvers could be further improved by optimizing the single processor performance of major loops.

The robustness of our multigrid solvers with respect to increasing heterogeneity might be enhanced by using operator based interpolation and semicoarsening. That is, for increasing heterogeneity the number of V-cycles that is required for convergence will not change appreciably.

In this work, we have demonstrated that by combining powerful algorithms and current generation high performance computers, the capabilities of computer models for groundwater modeling can be substantially enhanced.

REFERENCES

1. G. Mahinthakumar and A.J. Valocchi, Application of the connection machine to flow and transport problems in three-dimensional heterogeneous aquifers, *Advances in Water Resources* **15**, 289–302, (1992).
2. P.D. Meyer, A.J. Valocchi, S.F. Ashby and P.E. Saylor, A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media, *Water Resources Research* **25**, 1440–1446, (1989).
3. R.E. Alcouffe, A.Brandt, J.E. Dendy, Jr. and J.W. Painter, The multi-grid method for the diffusion equation with strongly discontinuously coefficients, *SIAM J. Sci. Stat. Comput.* **2**, 430–454, (1981).
4. A. Behie and P. Forsyth, Jr., Multigrid solution of three-dimensional problems with discontinuous coefficients, *Appl. Math. Comp. Soc. Petr. Eng. J.* **13**, 229–240, (1983).
5. T.J. McKeon and W.-C. Chu, A multigrid model for steady flow in partially saturated porous media, *Water Resources Research* **23**, 542–550, (1987).
6. S.F. Ashby and R.D. Falgout, A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations, Technical Report UCRL-JC-122359, Lawrence Livermore National Laboratory, CA, (October, 1995).
7. G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd edition, Johns Hopkins University Press, Baltimore, MD, (1989).
8. H.A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric problems, *SIAM J. Sci. Stat. Comput.* **13**, 631–645, (1992).

9. R. Barrett *et al.*, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM Publications, Philadelphia, PA, (1993).
10. A. Brandt, Multilevel adaptive solutions to boundary-value problems, *Math. Comp.* **31**, 311–329, (1977).
11. W.L. Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, PA, (1988).
12. M. Holst and F. Saied, Multigrid solution of the Poisson-Boltzmann equation, *J. Comp. Chem.* **14** (1), 105–113, (1993).
13. M. Holst, R. Kozack, F. Saied and S. Subramaniam, Treatment of electrostatic effects in proteins: Multigrid-based Newton iterative method for solution of the full nonlinear Poisson-Boltzmann equation, *Proteins: Structure, Function, and Genetics* **18** (3), 231–245, (1994).
14. J.D. Istok, *Groundwater Modeling by the Finite Element Method*, Water Resources Monograph 13, American Geophysical Union, Washington, DC, (1989).
15. P.S. Huyakorn and G.F. Pinder, *Computational Methods in Subsurface Flow*, Academic Press, New York, (1983).
16. G. Mahinthakumar and F. Saied, Multigrid and Krylov solvers for large scale finite element groundwater flow simulations on distributed memory parallel platforms, Technical Report ORNL/TM-13441, Oak Ridge National Laboratory, TN, (July 1997) (also available from `http://www.ccs.ornl.gov/staff/kumar/solver.html`).